

Coding Styles Ensimag 3A SLE

Matthieu Moy

Matthieu.Moy@imag.fr

2015-2016



Sommaire

- 1 Dans la vie courante ...
- 2 Code : fond et forme
- 3 Directives de codage du noyau Linux

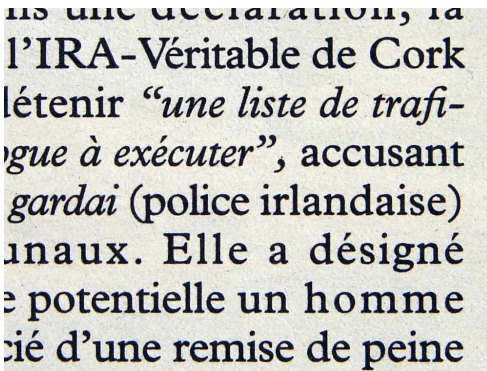
Une page de journal ...



Regardons d'un peu plus près ...



Et encore un peu plus près ...



Une page de journal ...

- Page découpée en plusieurs articles,
- Chaque article a un titre,
- Les articles longs sont découpés avec des sous-titres.



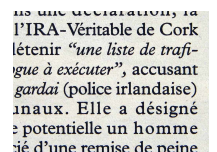
Regardons d'un peu plus près ...

- Disposition en colonnes (≈ 40 caractères de large)
⇒ lecture plus facile (retour des yeux à la ligne suivante).
- Découpage du texte en paragraphe
⇒ Saut de paragraphe = petite pause, passage à une autre notion.



Et encore un peu plus près ...

- Placement des espaces et ponctuation
⇒ lecture fluide.
 - ▶ Pas d'espaces à l'intérieur des parenthèses
 - ▶ Espaces autour des parenthèses
 - ▶ Espace après une virgule mais pas avant
 - ▶ ...



Et si on faisait autrement ?

Tout informaticien doit connaître le langage C.C ' est une espèce d 'espéranto de l'informatique .Les autres langages fournissent en effet souvent une interface avec C(ce qui leur permet en particulier de s'interfacer plus facilement avec le système d ' exploitation) ou sont eux-mêmes écrits en C.D'autre part c ' est le langage de base pour programmer les couches basses des systèmes informatiques. Par exemple, on écrit rarement un pilote de périphérique en Ada ou Java. Finalement ,en compilation, C est souvent choisi comme cible de langages de plus haut niveau. Toutefois, il est peu probable(ou plus exactement ,peu souhaitable) qu'un ingénieur informaticien soit confronté à de gros développements logiciels entièrement en C.L'objectif pédagogique du projet est donc surtout de montrer comment C peut servir d ' interface entre les langages de haut niveau et les couches basses de la machine.Plus précisément, les objectifs du stage C de première année sont : Apprentissage de C(en soi ,et pour la démarche qui consiste à apprendre un nouveau langage). Lien du logiciel avec les couches basses de l'informatique ,ici logiciel de base et architecture. Le premier projet logiciel un peu conséquent ,à développer dans les règles de l ' art(mise en œuvre de tests ,documentation, démonstration du logiciel, partage du travail, ...)Lien avec les autres modules de première année(théorie des langages) et de deuxième année (compilation, système)

Tentons de faire mieux ...

Tout informaticien doit connaître le langage C. C'est une espèce d'espéranto de l'informatique. Les autres langages fournissent en effet souvent une interface avec C (ce qui leur permet en particulier de s'interfacer plus facilement avec le système d'exploitation) ou sont eux-mêmes écrits en C. D'autre part c'est le langage de base pour programmer les couches basses des systèmes informatiques. Par exemple, on écrit rarement un pilote de périphérique en Ada ou Java. Finalement, en compilation, C est souvent choisi comme cible de langages de plus haut niveau. Toutefois, il est peu probable (ou plus exactement, peu souhaitable) qu'un ingénieur informaticien soit confronté à de gros développements logiciels entièrement en C. L'objectif pédagogique du projet est donc surtout de montrer comment C peut servir d'interface entre les langages de haut niveau et les couches basses de la machine. Plus précisément, les objectifs du stage C de première année sont : Apprentissage de C (en soi, et pour la démarche qui consiste à apprendre un nouveau langage). Lien du logiciel avec les couches basses de l'informatique, ici logiciel de base et architecture. Le premier projet logiciel un peu conséquent, à développer dans les règles de l'art (mise en œuvre de tests, documentation, démonstration du logiciel, partage du travail, ...) Lien avec les autres modules de première année (théorie des langages) et de deuxième année (compilation, système)

Tentons de faire mieux ...

Tout informaticien doit connaître le langage C. C'est une espèce d'espéranto de l'informatique. Les autres langages fournissent en effet souvent une interface avec C (ce qui leur permet en particulier de s'interfacer plus facilement avec le système d'exploitation) ou sont eux-mêmes écrits en C. D'autre part c'est le langage de base pour programmer les couches basses des systèmes informatiques. Par exemple, on écrit rarement un pilote de périphérique en Ada ou Java. Finalement, en compilation, C est souvent choisi comme cible de langages de plus haut niveau. Toutefois, il est peu probable (ou plus exactement, peu souhaitable) qu'un ingénieur informaticien soit confronté à de gros développements logiciels entièrement en C. L'objectif pédagogique du projet est donc surtout de montrer comment C peut servir d'interface entre les langages de haut niveau et les couches basses de la machine. Plus précisément, les objectifs du stage C de première année sont : Apprentissage de C (en soi, et pour la démarche qui consiste à apprendre un nouveau langage). Lien du logiciel avec les couches basses de l'informatique, ici logiciel de base et architecture. Le premier projet logiciel un peu conséquent, à développer dans les règles de l'art (mise en œuvre de tests, documentation, démonstration du logiciel, partage du travail, ...) Lien avec les autres modules de première année (théorie des langages) et de deuxième année (compilation, système)

Tentons de faire mieux ...

Tout informaticien doit connaître le langage C. C'est une espèce d'espéranto de l'informatique. Les autres langages fournissent en effet souvent une interface avec C (ce qui leur permet en particulier de s'interfacer plus facilement avec le système d'exploitation) ou sont eux-mêmes écrits en C. D'autre part c'est le langage de base pour programmer les couches basses des systèmes informatiques. Par exemple, on écrit rarement un pilote de périphérique en Ada ou Java. Finalement, en compilation, C est souvent choisi comme cible de langages de plus haut niveau. Toutefois, il est peu probable (ou plus exactement, peu souhaitable) qu'un ingénieur informaticien soit confronté à de gros développements logiciels entièrement en C. L'objectif pédagogique du projet est donc surtout de montrer comment C peut servir d'interface entre les langages de haut niveau et les couches basses de la machine. Plus précisément, les objectifs du stage C de première année sont :

- Apprentissage de C (en soi, et pour la démarche qui consiste à apprendre un nouveau langage).
- Lien du logiciel avec les couches basses de l'informatique, ici logiciel de base et architecture.
- Le premier projet logiciel un peu conséquent, à développer dans les règles de l'art (mise en œuvre de tests, documentation, démonstration du logiciel, partage du travail, ...)
- Lien avec les autres modules de première année (théorie des langages) et de deuxième année (compilation, système)

Tentons de faire mieux ...

Tout informaticien doit connaître le langage C. C'est une espèce d'espéranto de l'informatique. Les autres langages fournissent en effet souvent une interface avec C (ce qui leur permet en particulier de s'interfacer plus facilement avec le système d'exploitation) ou sont eux-mêmes écrits en C. D'autre part c'est le langage de base pour programmer les couches basses des systèmes informatiques. Par exemple, on écrit rarement un pilote de périphérique en Ada ou Java. Finalement, en compilation, C est souvent choisi comme cible de langages de plus haut niveau.

Toutefois, il est peu probable (ou plus exactement, peu souhaitable) qu'un ingénieur informaticien soit confronté à de gros développements logiciels entièrement en C. L'objectif pédagogique du projet est donc surtout de montrer comment C peut servir d'interface entre les langages de haut niveau et les couches basses de la machine. Plus précisément, les objectifs du stage C de première année sont :

- Apprentissage de C (en soi, et pour la démarche qui consiste à apprendre un nouveau langage).
- Lien du logiciel avec les couches basses de l'informatique, ici logiciel de base et architecture.
- Le premier projet logiciel un peu conséquent, à développer dans les règles de l'art (mise en œuvre de tests, documentation, démonstration du logiciel, partage du travail, ...)
- Lien avec les autres modules de première année (théorie des langages) et de deuxième année (compilation, système)

Conclusion

- Le fond est important
- ... mais on a du mal à se concentrer sur le fond si la forme est imparfaite !

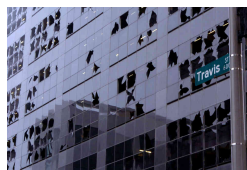
~~ C'est pareil pour du code !

Principe de la « fenêtre brisée »

- Une fenêtre brisée

⇒ le vent s'engouffre, des saletés entrent

⇒ « mon voisin n'entretient pas ses fenêtres, pourquoi je le ferais ? »



- Une ligne mal indentée

⇒ « je ne vais pas corriger, ça va faire des conflits dans Git »

⇒ « pourquoi je m'embêterais à formater mon code proprement ? »

⇒ « pourquoi j'écrirais de la doc, de toutes façons, le code est sale »

⇒ « la dernière fois que j'ai codé, je n'ai pas écrit de tests, ça s'est bien passé quand même ... »

⚠ Attention à la dette technique

Peut-on faire mieux ?

```
struct noeud_t{
    unsigned val;
    struct noeud_t *fg ;
    struct noeud_t * fd;
};

int rechercher(unsigned v,struct noeud_t *arbre)
{
    int res;
    if(arbre != NULL) {
        if (v==arbre->val){
            return 1;
        } else if(v<arbre->val){
            return rechercher(v,arbre->fg);
        }
        else
        {
            res =rechercher( v, arbre -> fd ) ;
            return res ;
        }
    } else {
        return 0 ;
    }
}
```

Peut-on faire mieux ?

```
struct noeud_t{
    unsigned val;
    struct noeud_t *fg ;
    struct noeud_t *fd;
};

int rechercher(unsigned v, struct noeud_t *arbre)
{
    int res;
    if(arbre != NULL) {
        if (v==arbre->val){
            return 1;
        } else if(v<arbre->val){
            return rechercher(v, arbre->fg);
        } else {
            res =rechercher( v, arbre -> fd ) ;
            return res ;
        }
    } else {
        return 0 ;
    }
}
```

Peut-on faire mieux ?

```
struct noeud_t {
    unsigned val;
    struct noeud_t *fg;
    struct noeud_t *fd;
};

int rechercher(unsigned v, struct noeud_t *arbre)
{
    int res;
    if (arbre != NULL) {
        if (v == arbre->val) {
            return 1;
        } else if (v < arbre->val) {
            return rechercher(v, arbre->fg);
        } else {
            res = rechercher(v, arbre->fd);
            return res ;
        }
    } else {
        return 0;
    }
}
```

Peut-on faire mieux ?

```
struct noeud_t {
    unsigned val;
    struct noeud_t *fg;
    struct noeud_t *fd;
};

int rechercher(unsigned v, struct noeud_t *arbre)
{
    int res;
    if (arbre == NULL) {
        return 0;
    } else {
        if (v == arbre->val) {
            return 1;
        } else if (v < arbre->val) {
            return rechercher(v, arbre->fg);
        } else {
            res = rechercher(v, arbre->fd);
            return res;
        }
    }
}
```

Peut-on faire mieux ?

```
struct noeud_t {
    unsigned val;
    struct noeud_t *fg;
    struct noeud_t *fd;
};

int rechercher(unsigned v, struct noeud_t *arbre)
{
    if (arbre == NULL) {
        return 0;
    } else {
        if (v == arbre->val) {
            return 1;
        } else if (v < arbre->val) {
            return rechercher(v, arbre->fg);
        } else {
            return rechercher(v, arbre->fd);
        }
    }
}
```

Peut-on faire mieux ?

```
struct noeud_t {
    unsigned val;
    struct noeud_t *fg;
    struct noeud_t *fd;
};

int rechercher(unsigned v, struct noeud_t *arbre)
{
    if (arbre == NULL) {
        return 0;
    } else if (v == arbre->val) {
        return 1;
    } else if (v < arbre->val) {
        return rechercher(v, arbre->fg);
    } else {
        return rechercher(v, arbre->fd);
    }
}
```

↪ on va s'arrêter là !

Directives de codage du noyau Linux

- Utilisées par Linux et beaucoup d'autres projets
- Dispo un peu partout sur le Web (<http://lxr.linux.no/#linux/Documentation/CodingStyle>)
- Bon ou mauvais style : question subjective, mais l'important est d'être homogènes !

"First off, I'd suggest printing out a copy of the GNU coding standards, and **NOT** read it. Burn them, it's a great symbolic gesture."

Indentation, présentation du code

- ❶ Indentation à 8 caractères : blocs if/while/for/... clairement visibles.
- ❷ Pas de ligne de plus de 80 caractère : plus lisible, affichable sur une petite fenêtre.
- ❸ Une fonction doit tenir sur un écran (en théorie, 1 écran = 25 lignes). Faire des fonctions courtes, qui font une chose et qui le font bien.
- ❹ Si on a l'impression qu'il y a conflit entre les deux premiers points, c'est qu'il faut mieux découper (cf. point 3).
- ❺ Utiliser les lignes vides pour marquer une pause (cf. notion de paragraphe en français)

Nommage des fonctions, variables

- Noms courts et **expressifs**
 - ▶ ThisVariableIsATemporaryCounter ⇒ trop long
 - ▶ a, var1 ⇒ pas expressifs
- Nom expressif indispensable pour les variables/fonctions globales.

Commentaires

- Expliquez **pourquoi** votre code est comme il est, et non **comment**.
- Si le code a besoin de beaucoup de commentaire pour expliquer comment il fonctionne, c'est qu'il est trop complexe \Rightarrow simplifiez-le plutôt que de le commenter !